An API for Integrating Spatial Context Models with Spatial Reasoning Algorithms

Mikkel Baun Kjærgaard
University of Aarhus
Department of Computer Science
Aabogade 34, DK-8200 Aarhus N
mikkelbk@daimi.au.dk

Abstract

The integration of context-aware applications with spatial context models is often done using a common query language. However, algorithms that estimate and reason about spatial context information can benefit from a tighter integration. An object-oriented API makes such integration possible and can help reduce the complexity of algorithms making them easier to maintain and develop. This paper propose an object-oriented API for context models of the physical environment and extensions to a location modeling approach called geometric space trees for it to provide adequate support for location modeling. The utility of the API is evaluated in several real-world cases from an indoor location system, and spans several types of spatial reasoning algorithms.

1 Introduction

Within the area of pervasive computing the great vision is moving computation from the desktop computer to a number of devices embedded in the environment of the user. The applications put forward for these devices depend on the notion of context-awareness. One ingredient when making context-aware applications is information concerning the physical environment in which the devices are situated.

In the past, several approaches to modeling the layout of the physical environment in terms of location information have been proposed. It has been generally accepted that a hybrid approach to location modeling is necessary which describes both geometric and symbolic information [7, 12]. Several papers have identified requirements for location models [7, 8]. The requirements in [7] were drawn up based on the authors' experience of doing location modeling in different technical settings. The following requirements, for what it should be possible to model with a location model, are listed in that paper. It should be possible to model the geometry of objects relative to different coordinate systems, symbolic information, relations between objects other than the geometric, and the description of objects in different levels of detail maybe by relating different sub models.

For integrating location models with context-aware applications many techniques propose a common language for storing and accessing location information [7, 12]. This is because they have as their main design criteria to share location information between applications, so the modeling effort can be minimized. An example of a query these languages could support is: *find the nearest printer which is less than 50 meters away from a specific location*. These techniques make it easier to develop context-aware applications which do not need to do any further processing, than what can be achieved through the common language. However, when developing algorithms that estimate and reason about spatial context information one can benefit from a tighter integration than offered by a common language.

This paper proposes an object-oriented API named the Java Location Modeling API (JLMA). This API makes a tighter integration possible and can thereby help reduce the complexity of algorithms making them easier to maintain and develop. In the setting of Java as programming environment the proposed API is also an improvement with respect to the current Java API for location modeling named JSR-179 [13]. JSR-179 was designed for cellular phones and is tied to one coordinate system. It is therefore not applicable in general for context-aware applications. The proposed API is based on geometric space trees (GSTs) [12]. However, this paper also propose extensions to GSTs so they satisfy the requirements in [7]. The design of the API, including the extensions to geometric space trees, is presented in Section 2. The implementation of the API is outlined in Section 3. The evaluation of the API is presented in Section

Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)
0-7695-2520-2/06 $20.00 © 2006 IEEE

4 and a discussion of the results and related work is given in Section 5.

2 Design

The design of the API can be viewed from two angles: First, from a perspective of location modeling and second, from a programming perspective.

2.1 Location modeling

As stated in the introduction, the design of this API has been based on the idea of GSTs [12], but these are not adequate if not extended. Basic GSTs are hierarchical structures of nodes where each node has a number of attributes associated with it. Figure 1 shows a layout of a floor and the nodes of a GST describing the floor. The attributes describes, among other things, the shape and extent of a node. For example, a node could have the shape of a polygon and the extent would then be a list of points describing the polygon. The attributes can also describe types associated with a node such as office, corridor, and building. Each node can also have attributes that define new coordinate systems for the subnodes of the node through a rotation matrix and an origin point. All nodes and shapes have a computable textual representation in the form of Area Location Identifiers[1] (ALIs). The necessary extensions to GSTs will be presented below structured by the requirements in [7].

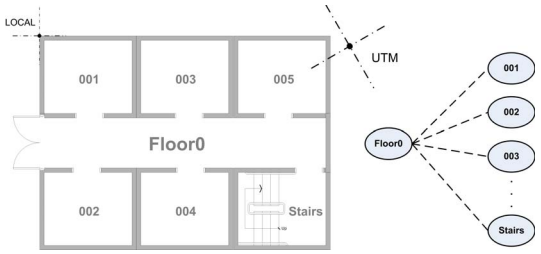

Figure 1. The layout of a floor in a building and the associated nodes of a GST.

2.1.1 Coordinate systems

GSTs as presented in [12] include hierarchies of coordinate systems where it is possible to translate objects between the coordinate systems. The ALIs basic GSTs directly support can denote nodes, areas and points. Therefore to satisfy the requirements in [7] the following usage of GSTs is necessary. GSTs should not be constrained to only two types of geometric forms and local coordinate systems. For example to work with the coordinates used in JSR-179 as part of a GST, a new point type using latitude, longitude and altitude as basic parameters would be needed. For converting to and from global coordinate systems such as WGS84, which is used by the JSR-179, new translations would be needed. These translations cannot be defined using only a rotation matrix and an origin point but more general translations should be available for making projections to, for instance, WGS84. Figure 1 shows, in addition to a local coordinate system, a Universal Transverse Mercator (UTM) projection which the local coordinate system should be able to be mapped to. When new types of geometric forms are added, the syntax of ALIs need to be extended with a textual form of each new geometric object type. For instance to describe a line representing a walking path in the building shown on Figur 1, the following ALI could be used `ali://floor0#{(50.0,350.0,0.0),(520.0,350,0.0),(470.0,550.0,0.0)}`. Another example of geometric objects types which could be added to the API, are the types defined in the Simple Features Specification (SFS) from the Open GIS Consortium [14].

2.1.2 Symbolic information

GSTs as presented in [12] include symbolic information such as names, types, and the hierarchical relations between them. To satisfy the requirements in [7], one new type of symbolic information should be added. This is descriptive information such as for example the address information included in JSR-179 [13] or the units used by a coordinate system of a node. A key-value mechanism for each node could be added for representing this type of information. To model address information a number of key-value pairs such as: `addressinfo.street=Aabogade 142` could be used.

2.1.3 Relations between objects

GSTs as described in [12] only have the hierarchical structure for modeling relations between objects other than the geometric. In [11] GSTs were extended to include information on how nodes are connected. In this API, the extension of GSTs with relations is done by having a direct representation for connections and using the key-value mechanism for other relations. The reason why the key-value mechanism is seen as appropriate for other relations is that since all nodes can be referenced using ALIs, it is straightforward to represent relations between nodes.

[1] The author has renamed the first part of the acronym from Aura to area for generality.

Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)
0-7695-2520-2/06 $20.00 © 2006 IEEE

2.1.4 Level of detail

For representing objects at different levels of detail GSTs provide again only the hierarchical structure of nodes. This is only appropriate for self-contained models, but it does not give the option of relating different models. Therefore GSTs should be extended to include relations across hierarchies. The extension of GST with a submodel relation can be done as described above for general relations. There is, however, one extra issue in terms of translating between the coordinate systems of the sub- and supernodes. Therefore it should be possible to define translations between different hierarchies. These translations are in this work constrained only to be defined between root nodes for limiting the complexity of the translations which the implementation of the API has to consider.

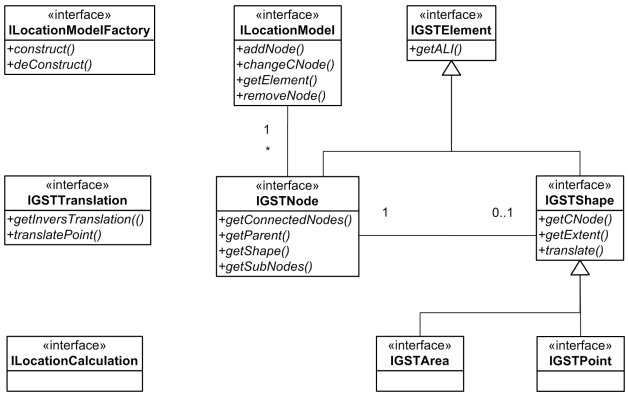

Figure 2. UML Diagram of the API

2.2 Programming

The API has been designed to enable location modeling using GSTs with the extensions described above. To enable this the following elements have to be represented in the API:

- **GST nodes:** The hierarchical nodes of the trees. Each of these nodes has: a parent node associated with it if it is not a root node, a set of subnodes if it is not a leaf node, a set of types, a set of nodes it is connected to, key-value properties, translations, and a geometric object describing the shape and extent.
- **Geometric objects:** The geometric objects used to describe locations of interest. Each element has some way of denoting its shape and extent and a reference to the node which denotes the coordinate system the extent is defined in.
- **Translations:** The translations between different coordinate systems. Each translation can translate a point from one coordinate system to another and calculate the inverse translation.

Figure 2 shows an UML diagram of the API including the central methods of the interfaces. In the API nodes are represented by the interface `IGSTNode`. It defines methods for handling the information associated with a node as described above. The Geometric objects are represented by the interface `IGSTShape` which denotes that implementing objects have a geometric shape and extent. The interface has methods for getting the data which defines the shape and extent and for getting the node which represents the coordinate system. The API provides two specialized interfaces `IGSTPoint` and `IGSTArea` of `IGSTShape` each respectively representing the point and areas included in basic GSTs. The `IGSTShape` and `IGSTNode` both extend the common interface `IGSTElement` which represents entities which can be described using an ALI and therefore has methods to return an ALI describing the object. Translations are represented by the interface `IGSTTranslation` which has a method for translating a point between the different coordinate systems. It is then the responsibility of the single implementers of the IGSTShape interface to translate their geometric extent using this method.

The API also includes an interface `ILocationModel` which represents the model as a whole. It provides methods for querying the model for objects representing ALIs, adding and removing nodes, and translating implementers of the `IGSTShape` interface between different coordinate systems. For instance to translate an object to a new coordinate system the method `changeCNode(IGSTShape shape,IGSTNode coornode)` has to be invoked. This method will return a new shape object in the coordinate system represented by `coornode`. This makes it easy to translate a shape object to a different coordinate system.

One thing to note is that objects implementing `IGSTShape` should be handled as value objects, meaning that when converting an object implementing `IGSTShape` to a new coordinate system a new object should be created. This removes the chance of algorithms becoming unstable resulting from updates to objects used in more than one context. In connection with the model, the technical concern of storage and representation of nodes and spaces also needs to be addressed. Here the `ILocationModelFactory` defines methods for constructing and deconstructing models which makes it possible for the implementer to choose the best solution for the specific implementation. The interface `IGSTCalculation` represents the calculations which can be preformed on the models. A more detailed description of the API is available from [3].

Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)
0-7695-2520-2/06 $20.00 © 2006 IEEE

3 Implementation

The proposed API has been constructed using Java interfaces as described in the previous section. A reference implementation of these interfaces has been implemented using J2SE 1.4. The interfaces and the implementation including documentation are available from [3]. The reference implementation implements all the interfaces of the API. Three types of translations have been implemented. The first type is the translation type described in [12] that can be used when translating between different local coordinate system and UTM. The second type is for translating between UTM and WGS84. This implementation is based on the package GeoTransfrom [1]. The third translation is a pass through translation for representing a node that does not specialize the coordinate system of its parent node. For all of the different coordinate systems the same point type is used but with different interpretations based on the associated coordinate system. To implement the method `changeCNode` which translate between coordinate systems an algorithm has been designed that calculates the translation path between the old and the new coordinate system. The algorithm is as follows: If the coordinate system nodes are in the same GST then the method `getCommonAncestor` of `ILocationModel` is used to find the path from the old coordinate system to a common ancestor and then the path from this node to the new coordinate system node is found. If the coordinate systems are not in the same GST the path from the old coordinate system to the root of the GST where the old coordinate system is situated is found. From here the path to the GST of the new coordinate system is found using the root translations available. Finally, the path from the root node of the GST where the new coordinate system node is situated to the new coordinate system node is found. Several implementations of the `IGSTCalculation` interface has also been made which can calculate for instance shortest path, distance, within, and contains in relation to nodes and shapes. For storage an XML format has been specified using a XML Schema also available from [3]. A version of the `ILocationModelFactory` interface which can construct and deconstruct models from this format has been implemented. The reference implementation depends on two other packages. For matrix calculation it uses the Java Matrix Package (JAMA) [2] and for handling XML it uses the JDOM package [4].

4 Evaluation

The design and implementation of the API has been carried out in the context of a project building an indoor location system for wireless DECT phones. The system build in the project has also been used to evaluate the utility of the API as presented in Sections 4.1-4.5. A performance evaluation of the reference implementation is given in Section 4.6. The utility evaluation is formed as cases from the system where the API has been used, and focuses therefore mostly on the usage of the API. For each case, the usage of the API will be presented and the APIs ability to support the case will be discussed. To ensure that the cases span a wide range of usages of the API, the selected cases from the system covers most of the layers defined as part of The Location Stack [9, 10]. This stack is considered state-of-the-art and has also been used in Intel's Place Lab [5] initiative. The layers covered in the evaluation are sensors, measurements, fusion, arrangements and contextual fusion. Two layers of the stack are not covered: the activities layer and the intentions layer, because they only deal with location information on a very abstract level.

4.1 Sensors

The sensor layer of the location stack includes sensor hardware and software drivers. The evaluation case from the project at this layer is: *The marking of calibration measurements with location information.* In the implemented location system sensors markes measurements using ALIs and these ALIs is then interpreted at a server using the API. This evaluation case shows that because the API has a textual representation in the form of ALIs it is possible to integrate with lower level sensors. Also the APIs possibility to look up objects representing the locations made it easy to add observations to an observation-store on a server.

4.2 Measurements

The measurements layer contains algorithms for processing raw sensor data into canonical measurement types. Two evaluation cases are presented at this layer: *The first is an indoor localization algorithm based on time measurements and the second is an algorithm based on signal strength measurements.* The implemented time based algorithm uses the API when looking up base station positions represented with ALIs and when transforming coordinates to a common coordinate system for further processing by various matrix calculations. The signal strength algorithm uses the API for associating observations and entries of a sensor model with locations.

These two evaluation cases illustrate the use of objects representing locations in algorithms which process location information. Positive things to note of the usage of the API are: In the first case the API reduces the algorithm code because coordinate system translations are handled by the API. In the second case because the algorithm can use the API's objects to represent locations the algorithm does not need to have an internal representation of them reducing the

Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)
0-7695-2520-2/06 $20.00 © 2006 IEEE

code which handles the location information. The coupling between the sensor model and the algorithm becomes very simple because the API's objects can be used to query the sensormodel reducing the code size. The use of objects also made it possible to make a good coupling between the algorithm and an observation-store which handles the gathered observations. Unification of nodes and shapes in the API through the `IGSTElement` interface also made it easy to integrate the results from the two types of algorithms in further processing.

4.3 Fusion

The fusion layer contains algorithms which merge streams of data. The evaluation case at this layer is: *A tracking algorithm for indoor localization*. In the implemented algorithm the API is used for looking up nodes and the nodes type and connectivity information. This case shows the need for types and connectivity information which makes it possible to construct a good indoor tracking algorithm.

4.4 Arrangements

The arrangements layer contains algorithms which reason about the relationship between objects. The evaluation case for this layer is: *An algorithm which calculates the direct distance between two cells in terms of the number of separating cells. A cell is here a part of a room or a whole room depending on its size.* The implemented algorithm uses the API for looking up nodes and the nodes shape information. For this algorithm the API only provided limited support. Because there are a high number of potential cells the algorithm has to consider it has in practice to look up all nodes which are cells in the used model. So for this algorithm some query based interface for asking spatial queries on cells might be more appropriate.

4.5 Contextual Fusion

The contextual fusion layer contains systems which merge location data with other non-location contextual information such as personal data. The evaluation case at this layer is: *The integration of JLMA with JCAF [6] which is an API for context awareness.* The integration is based on the principle that only the JLMA objects which are necessary to describe the current context should be loaded into JCAF. In JCAF the interface `ContextItem` should be implemented by all objects which are used to describe context information. To integrate the JLMA objects with this interface two wrapper classes has been defined. The first for wrapping ALIs and the second for wrapping implementors of `IGSTElement`. In JCAF transformers are the mechanism used for the transformation of context information. For adding the ability to JCAF to look up ALIs and changing the coordinate system of shapes two transformers was implemented. The first transformer takes an ALI wrapper and transforms it into an JLMA object wrapper by doing a lookup of the ALI in the associated JLMA model. The second takes a shape and changes the coordinate system using the associated JLMA model. The implementation of the integration uses Java Remote Method Invocation (RMI) for communicating with JCAF. The use of RMI had the implication that JLMA objects in serializable versions had to be implemented and a remote interface for the JLMA model had to be defined. In this case JLMA was shown to be able to be integrated with a system for context fusion. The API provided means for modeling location aspects of context information in JCAF. Because the API included the `ILocationModel` interface for access to JLMA models it was possible to define a remote interface which made it possible to integrate JCAF with JLMA using RMI.

4.6 Performance

A performance evaluation of the reference implementation has also been carried out. The results are given in the table below. All performance evaluations were executed on a machine with Intel Pentium M 1.7GHz, 1 GB of memory, and running Windows XP. The first evaluation measures the time it takes to translate a point between two nodes with different distances between the nodes in the GST. The second evaluation measures the time it takes to calculate the node, out of a set of nodes, that is nearest to a specific point. The third evaluation measures the memory consumption of the reference implementation when loaded with GSTs of different sizes. In the first two evaluations the results are average values from 30,000 runs with random nodes selected from a model of the buildings containing the Department of Computer Science, University of Aarhus.

Nodes	2	3	4	4 (WGS-84)
Translation	0.49 ms	0.54 ms	0.58 ms	1.0 ms
Nodes	50	100	150	200
Nearest	1.1 ms	2.0 ms	3.1 ms	3.9 ms
Nodes	250	500	750	1000
Memory Use	502 kb	627 kb	929 kb	1193 kb

The results shows in both the translation and the nearest case, that the algorithms behave linearly with respect to the number of nodes. In the last evaluation of the translation case where a WGS84 translation was included, there was a performance degradation compared to the evaluations with only basic GST translations. The reference implementation's use of memory also appeared linear in the number of nodes for the used model.

Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)
0-7695-2520-2/06 $20.00 © 2006 IEEE

5 Discussion

In the introduction it was motivated that algorithms, which estimate and reason about spatial context information, can benefit from a tighter integration with spatial models using an object-oriented API. Several examples of these types of algorithms were given in the utility evaluation. In all cases except one there was a positive experience of using the object-oriented API. However, in the case of a distance reasoning algorithm the positive effect could not be noted. The reason was that in this case a large number of comparisons between locations had to be performed which might could be easier handled with a query. For the current reference implementation several improvements are possible in respect to performance and use of memory. The first is a caching strategy for translations and second an on demand loading strategy for the nodes of the location model.

5.1 Related work

The query interface designed for GST in [12] is, as stated earlier, appropriate when integrating with context-aware applications that depend on different spatial queries. In [12] a tighter integration of GSTs with a programming language is envisioned, and here the JLMA API could fulfill this role. In the Nexus project [7] a query interface has also been designed for satisfying spatial queries using the Augmented World Querying Language. A model for location information which is named the Augmented World Modeling Language has also been designed. This work was also as GSTs designed for integration with applications. In [15] a library for managing spatial context using arbitrary coordinate systems is presented. Compared to this API that work is more focused on managing conversions between arbitrary coordinate systems for supporting spatial queries and does not focus on the design of the API for such a library. One project which has focused more on algorithms processing location information is The Location Stack [9, 10]. However in that work, there has not been an explicit focus on how algorithms and location information is modeled and integrated. A grid based coordinate system is used that is implemented to solve the specific needs for the algorithms used. This solution is therefore not applicable in general.

6 Conclusion

An API for the integration of location models with algorithms processing location information was proposed. Extensions to Geometric Space Trees and the design of the API were presented. Results from evaluating the utility of API and the performance of the reference implementation were given and based on that the use of a query interface versus an object-oriented API was discussed.

References

[1] GeoTransform. http://www.geovrml.org/geotransform/.

[2] JAMA : A Java Matrix Package. http://math.nist.gov/javanumerics/jama/.

[3] Java Location Modeling API (JLMA). http://www.daimi.au.dk/∼mikkelbk/jlma.

[4] JDOM. http://www.jdom.org.

[5] Place Lab. http://www.placelab.org/.

[6] J. E. Bardram. The java context awareness framework (JCAF) - a service infrastructure and programming framework for context-aware applications. In *Lecture Notes in Computer Science*, volume 3468, 2005.

[7] M. Bauer, C. Becker, and K. Rothermel. Location models from the perspective of context-aware applications and mobile ad hoc networks. *Personal and Ubiquitous Computing*, 6(5/6):322–328, 2002.

[8] C. Becker and F. Dürr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9(1):20–31, 2005.

[9] D. Graumann, W. Lara, J. Hightower, and G. Borriello. Real-world implementation of the location stack: The universal location framework. In *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2003)*, pages 122–128, October 2003.

[10] J. Hightower, B. Brumitt, and G. Borriello. The location stack: A layered model for location in ubiquitous computing. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, pages 22–28, June 2002.

[11] W.-T. Hsieh, N. Miller, Y.-F. Yang, S.-C. Chou, and P. Steenkiste. Calculating walking distance in a hybrid space model. In *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning And Management (UbiComp 2004)*, September 2004.

[12] C. Jiang and P. Steenkiste. A hybrid location model with a computable location identifier for ubiquitous computing. In *Proceedings of the Fourth International Conference on Ubiquitous Computing (UBICOMP 2002)*, September 2002.

[13] K. Loytana. JSR-000179 Location API for J2ME. Java Community Process, http://www.jcp.org/en/jsr/detail?id=179, Last accessed August 2005, 2003.

[14] Open GIS Consortium. Simple Features Specification for SQL. http://www.opengis.org/techno/specs/99-049.pdf, Version 1.1 1999.

[15] T. Schwarz, N. Hönle, M. Grossmann, and D. Nicklas. A library for managing spatial context using arbitrary coordinate systems. In *PerCom Workshops 2004*, pages 48–54, 2004.

Acknowledgements

The research reported in this paper was partially funded by the software part of the ISIS Katrinebjerg competency centre http://www.isis.alexandra.dk/software/. Carsten Valdemar Munk helped implementing the reference implementation of the API.

Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)
0-7695-2520-2/06 $20.00 © 2006 IEEE